PATENT ABSTRACTS OF JAPAN

(11)Publication number : 11-003237
(43)Date of publication of application : 06.01.1999

(51)Int.Cl. G06F 9/46
G06F 3/14
G06F 15/00

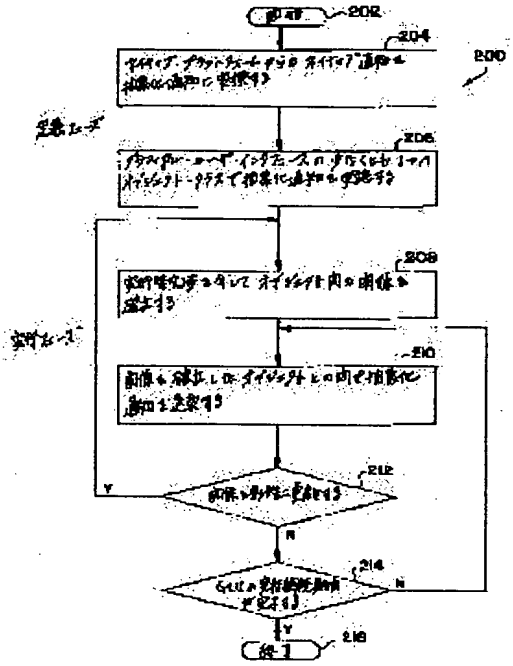
(21)Application number : 09-204333
(22)Date of filing : 30.07.1997

(71)Applicant : SUN MICROSYST INC
(72)Inventor : CABLE LAURENCE P G

(30)Priority
Priority number : 96 681917 Priority date : 30.07.1996 Priority country : US

(54) METHOD AND DEVICE FOR IMPROVING TRANSPORTABILITY OF OBJECT-ORIENTED INTERFACE AMONG PLURAL PLATFORMS

(57)Abstract:
PROBLEM TO BE SOLVED: To host again or transport an object-oriented graphical user interface processing system to another environment by abstracting the notifications which are performed in a native environment as the operating specifications.
SOLUTION: One or plural native notifications, i.e., events, state changes or interests which are carried out in regard to a 1st window base platform, i.e., a native platform are converted into the abstracted notifications which are used for defining a graphical user interface (204). Then the notifications are performed as the operating specifications via an optional object processing system (206). The relation is dynamically established at least to a single object via the verification of adaptability when an application is carried out (206). The objects can freely send and receive the notifications after the relation is established between the objects (210).



LEGAL STATUS

[Date of request for examination]

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

[Date of registration]

[Number of appeal against examiner's decision of rejection]

[Date of requesting appeal against examiner's decision of rejection]

[Date of extinction of right]

(19)日本国特許庁（ＪＰ）　(12)公開特許公報（Ａ）　(11)特許出願公開番号

特開平11－3237

(43)公開日　平成11年(1999)１月６日

(51)Int.Cl.[6]		識別記号	FI		
Ｇ０６Ｆ	9/46	３６０	Ｇ０６Ｆ	9/46	３６０Ｂ
	3/14	３４０		3/14	３４０Ａ
	15/00	３１０		15/00	３１０Ｒ

審査請求　未請求　請求項の数８　ＯＬ　（全 24 頁）

(21)出願番号　特願平9－204333

(22)出願日　平成９年(1997)７月30日

(31)優先権主張番号　０８／６８１９１７

(32)優先日　1996年７月30日

(33)優先権主張国　米国（ＵＳ）

(71)出願人　591064003
サン・マイクロシステムズ・インコーポレーテッド
SUN MICROSYSTEMS, INCORPORATED
アメリカ合衆国　94303　カリフォルニア州・パロ　アルト・サン　アントニオ　ロード・901

(72)発明者　ローレンス・ピイ・ジイ・ケーブル
アメリカ合衆国・94041・カリフォルニア州・マウンテン　ヴュー・ヴェラード　ストリート・295

(74)代理人　弁理士　山川　政樹

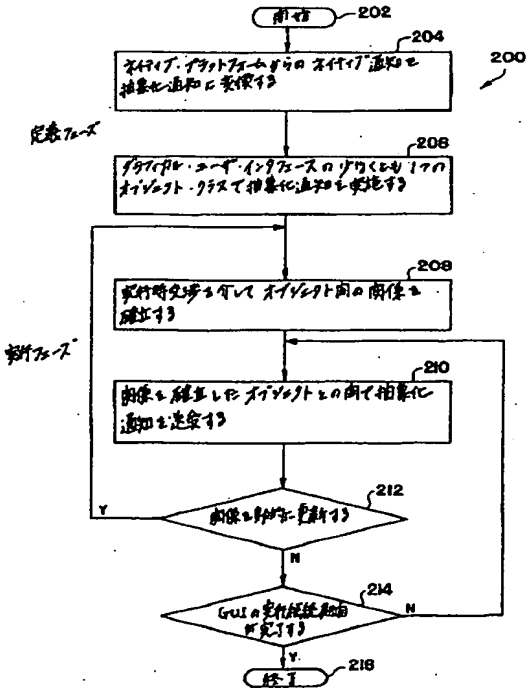
最終頁に続く

(54)【発明の名称】　複数のプラットフォーム間でのオブジェクト指向インタフェースの移植性を向上させる方法および装置

(57)【要約】

【課題】　オブジェクト指向グラフィカル・ユーザ・インタフェース処理系をあるネイティブ・ウィンドウ・ベース・プラットフォームまたは環境から他のウィンドウ・ベース・プラットフォームまたは環境に再ホストまたは移植できるようにする。

【解決手段】　ネイティブ環境で行われる通知（たとえば、事象や、状態変化や、「関心」）を動作仕様として抽象化する。この動作仕様は、グラフィカル・ユーザ・インタフェースの実行継続期間中に、特定のクライアント側オブジェクトが、異なるオブジェクトに関連付けられたサーバ側事象から抽象化された動作仕様に適合しているかどうかを判定するために、適合性交渉の一部として使用することができる。適合性交渉が成功した場合、事象のネイティブ処理系を使用するのではなく、抽象化通知がオブジェクトの特定のインスタンス間を流れシステムの状態をモデル化する。



1

【特許請求の範囲】

【請求項1】　グラフィカル・ユーザ・インタフェースのツールキットをウィンドウ・ベースのプラットフォームに移植する方法であって、

ウィンドウ・ベースのプラットフォームから状態変化のネイティブ通知を受取るステップと、

グラフィカル・ユーザ・インタフェースの実行時に、前記ネイティブ通知を、前記ウィンドウ・ベースのプラットフォームに特有の処理系とは独立のネイティブ通知の動作仕様を構成する抽象化通知として表すステップとを含むことを特徴とする方法。

【請求項2】　さらに、前記グラフィカル・ユーザ・インタフェースの実行時に、グラフィカル・ユーザ・インタフェース・ツールキット内の少なくとも１つのオブジェクトと前記動作仕様との適合性を検証するように前記グラフィカル・ユーザ・インタフェースを構成するステップを含むことを特徴とする請求項1に記載の方法。

【請求項3】　ウィンドウ・ベースのプラットフォームと共に使用できるグラフィカル・ユーザ・インタフェース・ツールキットであって、

ウィンドウ・ベースのプラットフォームから状態変化のネイティブ通知を受取る入力と、

前記ウィンドウ・ベースのプラットフォームに特有の処理系とは独立のネイティブ通知の動作仕様を構成する抽象化通知として表された前記ネイティブ通知の抽象を用いて関係を確立するオブジェクトの階層集合とを備えるグラフィカル・ユーザ・インタフェース・ツールキット。

【請求項4】　さらに、

前記抽象化通知をウィンドウ・ベースのプラットフォームへ送信する出力を備えることを特徴とする請求項3に記載のグラフィカル・ユーザ・インタフェース・ツールキット。

【請求項5】　グラフィカル・ユーザ・インタフェースのツールキットをウィンドウ・ベースのプラットフォームに移植する方法を実行するために使用される命令のプログラムをコード化するコンピュータ読取り可能な媒体であって、

ウィンドウ・ベースのプラットフォームから状態変化のネイティブ通知を受取るステップと、

グラフィカル・ユーザ・インタフェースの実行時に、前記ネイティブ通知を、前記ウィンドウ・ベースのプラットフォームに特有の処理系とは独立のネイティブ通知の動作仕様を構成する抽象化通知として表すステップとを含むプログラムを記録したことを特徴とするコンピュータ読取り可能な媒体。

【請求項6】　さらに、

グラフィカル・ユーザ・インタフェースの少なくとも１つのオブジェクト・クラスに抽象化通知を含めるステップを実施するプログラムを記録したことを特徴とする請求項5に記載のコンピュータ読取り可能な媒体。

2

【請求項7】　さらに、

前記グラフィカル・ユーザ・インタフェースの実行時に、グラフィカル・ユーザ・インタフェース・ツールキット内の少なくとも１つのオブジェクトと前記動作仕様との適合性を検証するステップを実施するプログラムを記録したことを特徴とする請求項5に記載のコンピュータ読取り可能な媒体。

【請求項8】　グラフィカル・ユーザ・インタフェースのツールキットをウィンドウ・ベースのプラットフォームに移植するコンピュータ・プログラムを用いてコード化されたコンピュータ・プログラム製品であって、

データを記憶するコンピュータ読取り可能な記憶媒体と、

コンピュータ読取り可能な記憶媒体に前記コンピュータ・コードを実行させるようにコンピュータ読取り可能な記憶媒体に記録されたコンピュータ・コード機構とを備え、そのコンピュータ・コード機構が、

ウィンドウ・ベースのプラットフォームから状態変化のネイティブ通知を受取る第1のコンピュータ・コード装置と、

グラフィカル・ユーザ・インタフェースの実行時に、前記ネイティブ通知を、前記ウィンドウ・ベースのプラットフォームに特有の処理系とは独立のネイティブ通知の動作仕様を構成する抽象化通知として表すために第1のコンピュータ・コード装置に結合された第2のコンピュータ・コード装置とを備えることを特徴とするコンピュータ・プログラム製品。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】本発明は、全般的には、ネットワーク・ベースのグラフィカル・ユーザ・インタフェースに関する。詳細には、本発明は、オブジェクト指向インタフェースを、様々な異なるハードウェア・プラットフォームまたはソフトウェア・プラットフォーム、あるいはその両方を統合するクライアント／サーバ環境と共に使用できるように、このインタフェースの複数のプラットフォーム間での移植性および保守容易性を向上させるシステムおよび方法に関する。

【０００２】

【従来の技術】既知のグラフィカル・ユーザ・インタフェースは、アプリケーション開発者に、スクロール・バー、プッシュ・ボタン、テキスト入力フィールドなどのオブジェクトの集合を含むアプリケーション・プログラム・インタフェース（ＡＰＩ）を備えている。オブジェクト指向グラフィカル・ユーザ・インタフェース（ＧＵＩ）は通常、これらのオブジェクトの階層化集合を「ツールキット」内に含んでいる。

【０００３】システム内のグラフィカル・ユーザ・インタフェースの動作は、クライアント／サーバ環境のクラ

3

イアントのオブジェクトとウィンドウ・ベースのシステム・サーバのオブジェクトとの間の対話として定義される。オブジェクト間の定義された対話を表すために使用されるパラダイムは、オブジェクト対話の「モデル・ビュー・コントローラ」（ＭＶＣ）パラダイムとも呼ばれる。モデル・ビュー・コントローラ・パラダイムは、システムの少なくとも一部（たとえば、システムのタイマ）内で状態の変化がどのように起こるかと、このような状態変化（クロックの遅延など）を観測することに対する関心を確立しているシステムの他の部分に、その変化をどのように伝達または反映するかを形式的に定義する。コントローラは、モデルに応答して実施される状態変化がどのようにビューに影響を及ぼすかを定義する１組の規則とみることができる。モデル・ビュー・コントローラ・パラダイムを使用すると、ツールキットは、グラフィカル・ユーザ・インタフェースを実施するモデル、ビュー、コントローラの階層化集合とみることができる。

【０００４】図１Ａは、モデル１０２とビュー１０４とコントローラ１０６との間の対話の抽象表現を示している。この場合、モデルは、記憶されているデータとみることができ、コントローラは、モデルとビューが対話するための規則とみることができる。図１Ａの矢印１０８は、最も一般的なモデル・ビュー・コントローラ・パラダイムを示し、この場合、状態変化は両方向に流れることができる。

【０００５】当業者には、図１Ｂを参照すると理解されるように、モデルとビューとコントローラが概念的には異なるものであっても、処理系においてモデルとビューとコントローラとの間の区別がなくなることが多い。たとえば、オブジェクト指向グラフィカル・ユーザ・インタフェースのスクロール・バー・オブジェクト１１２は、コントローラとビューの両方を表すことができる。図１Ｂのスクロール・バーは、たとえば、テキスト・ビュー１１６に表示されているテキスト・ファイル１１４の割合（すなわち、０％ないし１００％）をグラフィカルに表すときはビューである。スクロール・バーは、モニタ上のスクロール・バーの現在の位置を表示するためにマウスによって生成される一連の制御事象に応答する。本明細書では、「事象」は、１つのウィンドウ・ベース・プラットフォーム上の処理系に特有のものであり、移植性が必要なウィンドウ・ベースのプラットフォーム間では概念的に汎用である状態変化の情報を表す。

【０００６】この例では、スクロール・バー１１２はコントローラであってもよく、マウスの移動に応答してテキスト・ファイル１１４から得たテキストの表示の更新を開始または制御するために使用される。すなわち、スクロール・バーを「クリック」し、その画像１１２を垂直に移動させることによって、スクロール・バーはコントローラとして働き、スクロール・バーがディスプレイ

4

内を上下に移動するにつれてスクロール・バーを連続的に再描画し、それによってスクロール・バーの表示、したがって表示されるファイル１１４の内容を変更するために送信され、ビュー１１６によって使用される通知のストリームを生成する。図１Ｂの例では、モデル（すなわち、テキスト・ファイル１１４）は、マウスのユーザ作動が、スクロール・バー、したがってテキスト・ファイルの表示ビューにどのように影響を及ぼすかを定義する。

【０００７】Massachusetts Institute of Technologyから市販されているＵＮＩＸベースのＸ－Ｗｉｎｄｏｗ Ｓｙｓｔｅｍ™など通常のウィンドウ・ベースのシステムでは、システム内の状態変化は、上記でモデル・ビュー・コントローラ・パラダイムに関して説明した方式と同様な方式で「事象」を介してグラフィカル・ユーザ・インタフェース・クライアントに伝達される。ウィンドウ・ベースのシステムがオブジェクト指向グラフィカル・ユーザ・インタフェース・ツールキットに関連付けられると、ツールキットは、ウィンドウ・システム事象を介して報告されるウィンドウ・ベースのシステムの状態変化をオブジェクト指向モデル・ビュー・コントローラ上にマップする。ウィンドウ・ベースのシステムの事象によって、グラフィカル・ユーザ・インタフェースのオブジェクト上でメソッドが呼び出され、メソッドに状態変化が通知される。

【０００８】モデルとビューとコントローラとの間の関係の定義は、オブジェクト「クラス」の処理系特有の定義によって確立される。アプリケーションの継続期間中にモデル、ビュー、コントローラの特定のインスタンス間の関係を動的に指定することができるが、このような関係は、モデルおよびビューの特定の処理系のインスタンス間の割り当てを用いる場合を除いて、クライアント側アプリケーションの実行時に変更することはできない。これはたとえば、オブジェクト指向システムがＣ＋＋などのプログラミング言語で実施される場合にそうである。

【０００９】オブジェクト間の対話は、特定のウィンドウ・ベース環境と共に使用されるプラットフォーム特有の処理系として開発されるモデル・ビュー・コントローラによって定義されるので、ツールキットをあるウィンドウ・ベース・プラットフォームから他のプラットフォームに容易に移植することはできない。すなわち、あるツールキットが実施されたネイティブ・ウィンドウ・ベース・プラットフォームに存在する「事象」の処理系は、そのツールキットを移植する他のウィンドウ・ベース・プラットフォームと著しく異なることがある。さらに、ネイティブ処理系は、他の環境への移植性を目標として設計されるわけではなく、そのため、プラットフォーム依存性は通常、ネイティブ・プラットフォーム内ではローカライズまたは抽象化されない。あるプラットフ

5

ォームに関して定義される「事象」の処理系は通常、与えられる情報（すなわち、異なる意味）と、そのような情報を与えるために用いられるフォーマット（すなわち、異なる構文）の点で他のプラットフォームとは異なる。したがって、あるウィンドウ・ベース・プラットフォームに関して定義されたツールキットを他のウィンドウ・ベース・プラットフォームと共に使用するために改定しなければならないコンピュータ・コードの量は過度であり、実際的なものではない。そのため、オブジェクト指向グラフィカル・ユーザ・インタフェースの、あるネイティブ・ウィンドウ・ベース・プラットフォームから他のウィンドウ・ベース・プラットフォームへの移植（たとえば、ＮｅＸＴ　Ｃｏｍｐｕｔｅｒ　ＣｏｍｐａｎｙのＮｅＸＴＳＴＥＰ環境からＸＷｉｎｄｏｗ　Ｓｙｓｔｅｍ™など他のウィンドウ・ベース・プラットフォーム上への移植）には顕著な問題が存在する。

【００１０】

【発明が解決しようとする課題】したがって、プラットフォーム依存性をオブジェクト指向グラフィカル・ユーザ・インタフェースと共に使用できるようにローカライズし、抽象化する方法を開発することが望ましい。そこではオブジェクトの対話が抽象モデル・ビュー・コントローラに対して定義される。たとえば、オブジェクト指向グラフィカル・ユーザ・インタフェースのツールキットを表すために使用されるコードの主要な部分を変更できないようにし、したがって様々なウィンドウ・ベース・プラットフォームに容易に移植することができるように、ウィンドウ・ベースのシステムから受け取る事象を抽象化することが望ましい。

【００１１】

【課題を解決するための手段】本発明は、オブジェクト指向グラフィカル・ユーザ・インタフェース処理系をあるネイティブ・ウィンドウ・ベース・プラットフォームまたは環境から他のウィンドウ・ベース・プラットフォームまたは環境に再ホストまたは移植できるようにすることに関する。例示的な実施態様によれば、ネイティブ環境で行われる通知（たとえば、事象や、状態変化や、「関心」）は動作仕様として抽象化される。このような動作仕様（すなわち、トレイト（trait）またはプロトコル）は、グラフィカル・ユーザ・インタフェースの実行継続期間中に、特定のクライアント側オブジェクトが、異なるオブジェクトに関連付けられたサーバ側事象から抽象化された動作仕様に適合しているかどうかを判定するために、適合性交渉の一部として使用することができる。適合性交渉が成功した場合、事象のネイティブ処理系を使用するのではなく、抽象化通知がオブジェクトの特定のインスタンス間を流れ、システムの状態をモデル化することができる。実行継続期間中に、他のオブジェクトは、抽象化通知を受取るために、抽象化通知を含むオブジェクト・クラスとの関係を動的に確立することができる。

6

【００１２】本発明の例示的な実施態様は、オブジェクト指向グラフィカル・ユーザ・インタフェースのツールキット全体にわたってプラットフォーム処理系依存性を分離し、ローカライズする抽象化層をネイティブ処理系内に作成する。例示的な実施態様は、クライアント側モデル・ビュー・コントローラ・パラダイムと、ユーザ・インタフェース内のこのパラダイムのアプリケーションを著しく拡張し、移植タスクの規模を縮小し、いくつかの異なるウィンドウ・ベース・プラットフォーム間でのインタフェースの移植を可能にする。

【００１３】一般的に言えば、本発明の例示的な実施態様は、ウィンドウ・ベースのプラットフォームから状態変化のネイティブ通知を受取るステップと、グラフィカル・ユーザ・インタフェースの実行時に、前記ウィンドウ・ベースのプラットフォームに特有の処理系とは独立のネイティブ通知の動作仕様を構成する抽象化通知として前記ネイティブ通知を表すステップとを含む、グラフィカル・ユーザ・インタフェースのツールキットをウィンドウ・ベースのプラットフォームに移植する方法および装置に関する。本発明によれば、モデル、ビュー、コントローラの任意の処理系は任意に、抽象化通知に適合する。オブジェクト指向グラフィカル・ユーザ・インタフェース・ツールキットに定義されている抽象化通知は、グラフィカル・ユーザ・インタフェースのプログラミング言語を使用して実施することができる。したがって、グラフィカル・ユーザ・インタフェースの一部として確立されたツールキットは、様々なウィンドウ・ベース・システムを介して迅速にかつ費用有効に移植することができる。

【００１４】当業者には、本発明の前述およびその他の特徴が、本発明の下記の好ましい実施形態の説明を添付の図面に関連して読んだときに明らかになろう。

【００１５】

【発明の実施の形態】図2は、グラフィカル・ユーザ・インタフェースのツールキットの、ウィンドウ・ベースのプラットフォームへの（たとえば、第1のウィンドウ・ベース・プラットフォームから第2のウィンドウ・ベース・プラットフォームへの）移植に関する例示的なフローチャート２００を示す。図2のフローチャートの第1のフェーズは定義フェーズを構成し、このフェーズでは、ウィンドウ・ベースのプラットフォームの抽象化通知を使用してツールキットが定義される。図2のフローチャートの第2のフェーズは実行フェーズを構成し、このフェーズ中には、ウィンドウ・ベースのシステムからの通知が、ウィンドウ・ベースのシステムとグラフィカル・ユーザ・インタフェース・ツールキットとの間の対話を確立するために使用される抽象通知として表される。

【００１６】開始論理ブロック２０２ならびにステップ

7

２０４および２０６は定義フェーズを構成し、それに対して残りのステップは実行フェーズを構成する。図2のフローチャートのステップ２０４で、第1のウィンドウ・ベース・プラットフォーム（すなわち、ネイティブ・プラットフォーム）に関して実施される１つまたは複数のネイティブ通知（すなわち、事象または状態変化または「関心」）は、グラフィカル・ユーザ・インタフェースを定義する際に使用される抽象化通知に変換される。その結果、グラフィカル・ユーザ・インタフェース・ツールキットの各オブジェクトに、複数のウィンドウ・ベース・プラットフォームから受取った抽象化通知を登録することができる。

【００１７】当業者には理解されるように、通知は、キー・ストロークや、プッシュ・ボタンやアイコン化ウィンドウなどの活動化などの事象に対応させることができる。例示的な実施形態によれば、ホスト・ウィンドウ・ベースのプラットフォームから受取った、状態変化を示す事象は、オブジェクト・インタフェース定義の機能シグニチャとして表された抽象通知としてカプセル化される。抽象通知を表すために使用される機能シグニチャは、モデル、ビュー、コントローラ・オブジェクト処理系を適合させることのできる動作仕様（たとえば、フォーマル・トレイトまたはプロトコル）を構成する。

【００１８】モデル・ビュー・コントローラ・パラダイムはさらに、単一のモデルまたはコントローラに対する同じ関心を表す複数のビューをサポートし、複数のモデルまたはコントローラに対する関心を表すことのできる単一ビューをサポートするために拡張される。すなわち、グラフィカル・ユーザ・インタフェースの複数のオブジェクトに、複数のウィンドウ・ベース・プラットフォームから受取った抽象通知を登録することができる。

【００１９】定義フェーズのステップ２０６で、任意のオブジェクト処理系を使用して通知が動作仕様として実施される（たとえば、グラフィカル・ユーザ・インタフェースの処理系言語で形式的に表される）。すなわち、グラフィカル・ユーザ・インタフェースの少なくとも１つのオブジェクト・クラスで動作仕様を介して抽象化通知が実施される。しかし、抽象化通知は、オブジェクト指向グラフィカル・ユーザ・インタフェース・ツールキット内の任意の処理系のオブジェクト間で転送することができる。

【００２０】抽象化通知が、処理系言語によってサポートされる動作仕様として実施されるので、動作仕様に適合しているオブジェクトの任意のインスタンスは、グラフィカル・ユーザ・インタフェースの実行継続期間にわたって、同じ動作仕様に適合している他のオブジェクトとの任意の関係を動的に確立し、それによって移植性の高いシステムを形成することができる。具体的には、事象の機能シグニチャとしてのカプセル化を使用して、オブジェクト適合性検証が実施される。ネイティブ・ウィンドウ・ベース・システムの事象を、１つまたは複数のオブジェクトのインタフェース定義に対応する機能シグニチャとしてカプセル化することによって、オブジェクトが使用されるグラフィカル・ユーザ・インタフェースの実行継続期間中に、所与のオブジェクトとウィンドウ・ベースのシステムから受取った事象との適合性を検証することができる。

8

【００２１】定義フェーズの後に続いて、グラフィカル・ユーザ・インタフェース・ツールキットがウィンドウ・ベースのシステム（すなわち、ツールキットまたは他のウィンドウ・ベース・プラットフォームを実施するために使用されるネイティブ・プラットフォーム）と対話する実行フェーズを開始することができる。これはステップ２０８で表されており、このステップでは、アプリケーションの実行時に適合性検証を使用して少なくとも１つのオブジェクトと他のオブジェクトとの間の関係が動的に確立される。

【００２２】オブジェクト間の適合性は、オブジェクティブCなどのプログラミング言語が有するような動的（すなわち、実行時）結合機構を使用して、各オブジェクトと抽象化通知の適合性を検証することによって確立される。このような動的結合機構は、通知をネイティブ・ウィンドウ・ベース・プラットフォームの処理系言語で形式的に定義するために使用される。そのため、任意の通知に適合しているモデル、ビュー、コントローラの任意の処理系が与えられる。本発明によれば、適合性検証は、実行時交渉を任意のオブジェクトと共に使用して確立される。

【００２３】オブジェクト間に関係が確立された後、オブジェクトは、ステップ２１０で表したように自由に通知を送受することができる。当業者には理解されるように、任意の数の任意のオブジェクトに対して実行時交渉を実行し、それによってステップ２１２および２１４で表したように、オブジェクト指向グラフィカル・ユーザ・インタフェースの実行継続期間中に動的多面関係を確立し破壊することができる。

【００２４】したがって、例示的な実施形態によれば、グラフィカル・ユーザ・インタフェースのモデル・ビュー・コントローラ・パラダイムは、ビューが複数のモデルおよびコントローラにおけるいくつかの異なる関心を表すことができる「関心」モデルに拡張される。前述のステップは、グラフィカル・ユーザ・インタフェースの実行継続期間がステップ２１６の「終了」で完了するまで、任意の数の通知に対して、ステップ２１２および２１４に関連付けられたループを介して繰り返すことができる。

【００２５】例示的な実施形態によれば、オブジェクト指向グラフィカル・ユーザ・インタフェースは様々なウィンドウ・ベース・システムに容易にかつ動的に移植される。すなわち、グラフィカル・ユーザ・インタフェー

9

ス・ツールキットは、ツールキットをあるプラットフォームから他のプラットフォームに再ホストすることに関連する従来の難点を経験せずに、インタフェースが最初に実施されたネイティブ環境（たとえば、Ｍｉｃｒｏｓｏｆｔ™ Ｗｉｎｄｏｗｓ）から他のターゲット・ウィンドウ・ベース・プラットフォーム（たとえば、Ｘ Ｗｉｎｄｏｗ Ｓｙｓｔｅｍ™）に動的に移植することができる。

【００２６】例示的な実施形態によれば、ターゲット・プラットフォームに、直接対応する概念的特徴および処理系機能を有さない、ネイティブ・プラットフォームの概念的特徴および処理系機能に対するシステム依存性を除去するために抽象が使用される。当該の機能がグラフィカル・ユーザ・インタフェース・システムに基本的なものであり、最初の処理系内で広く使用されているので、対応する処理系機能のために、グラフィカル・ユーザ・インタフェースの従来型の再ホスティングは顕著な影響を受けていた。したがって、通常、最初に処理系コードを移植し、それを維持するのに必要な作業の量が多く、コストが非常に高かった。しかし、本発明の例示的な実施形態によれば、ネイティブ・プラットフォームとターゲット・プラットフォームとの間の顕著な差は、抽象化動作仕様を使用して、たとえばクライアント側に影響を与えるウィンドウ・ベースのシステムの状態変化をクライアント側に知らせることによって処理される。

【００２７】図3は、図2のフローチャートを実施する例示的なシステムを示す。例示的な図3のシステム３００は、サーバ側３０２と、クライアント側３０４と、キーボード入力やマウス入力などサーバ側への周辺入力３０６とを含む。通知は、プロセス間通信接続３０８を介してサーバ側のウィンドウ・ベース・プラットフォームからクライアント側のグラフィカル・ユーザ・インタフェースへ転送される。図3のシステムでは、キーボードの活動化、アイコン化ウィンドウの活動化、マウスの移動など、ウィンドウ・ベースのプラットフォームの事象は、クライアント側ツールキットによって抽象通知として表される（たとえば、変換される）。

【００２８】図2に関して説明したように、従来型のシステムとは異なり、ウィンドウ・ベースのシステムの事象は、グラフィカル・ユーザ・インタフェースのオブジェクトへは渡されない。その代わり、グラフィカル・ユーザ・インタフェースの各オブジェクトは最初、ウィンドウ・ベースのシステムから受取ることができる１つまたは複数の通知の抽象（すなわち、動作仕様）を備えた実行時交渉中に関心を登録しなければならない。図3の例では、グラフィカル・ユーザ・インタフェースの事象ディスパッチ・オブジェクトを使用して、抽象化通知がウィンドウ・ベースのプラットフォームから、実行時交渉中に特定の通知に関心を登録したグラフィカル・インタフェースの1つまたは複数の任意のオブジェクトにマップされる。

10

【００２９】動作時に、事象ディスパッチ・オブジェクトは、サーバ側のウィンドウ・ベースのプラットフォームから具体的な事象を受取ると、この事象を抽象通知にマップし、この抽象通知をグラフィカル・ユーザ・インタフェースの1つまたは複数のオブジェクトに渡す。その結果、最初にその通知に関心を持つオブジェクトとして登録したオブジェクトがその通知を受取る。従って、事象ディスパッチ・オブジェクトは、クライアント側オブジェクトに通知を渡す機構を構成する。

【００３０】当業者には理解されるように、前述の機能は、コンピュータ読取り可能な任意の媒体上で実施することができる。たとえば、そのような媒体は図3のグラフィカル・ユーザ・インタフェース３０４に存在することも、あるいはグラフィカル・ユーザ・インタフェース３０４と共に使用できるコンピュータ読取り可能な媒体に存在することもできる。この後者のケースでは、グラフィカル・ユーザ・インタフェースのツールキットをウィンドウ・ベースのプラットフォームに移植するコンピュータ・プログラムを用いてコンピュータ・プログラム製品３１０をコード化することができる。コンピュータ・プログラム製品はたとえば、ネイティブ・ウィンドウ・ベース・プラットフォームからの通知の抽象や、事象をターゲット・ウィンドウ・ベース・プラットフォームから様々な抽象化通知にマップするためのテーブルなどのデータを記憶するコンピュータ読取り可能な記憶媒体３１２を含むことができる。

【００３１】コンピュータ・プログラム製品３１０はさらに、コンピュータ読取り可能な記憶媒体にコンピュータ・コードを実行させるコンピュータ読取り可能な記憶媒体に埋め込まれたコンピュータ・コード機構を含むことができる。たとえば、コンピュータ・コード機構は、ウィンドウ・ベースのプラットフォームから状態変化のネイティブ通知を受取るように構成された第1のコンピュータ・コード装置３１４と、グラフィカル・ユーザ・インタフェースの実行時にネイティブ通知を抽象化通知として表すために第1のコンピュータ・コード装置に結合された第2のコンピュータ・コード装置３１６とを含むことができる。前述のように、ネイティブ通知は1つまたは複数の抽象化通知にマップすることができる。例示的な実施形態によれば、抽象化通知は、ネイティブ・ウィンドウ・ベース・プラットフォームとターゲット・ウィンドウ・ベース・プラットフォームの両方に特有の処理系とは独立のネイティブ通知の動作仕様を構成する。

【００３２】例示的な図3の実施形態では、コンピュータ・コード機構は任意の数のコンピュータ・コード装置を含むことができる。たとえば、グラフィカル・ユーザ・インタフェース・ツールキット内の少なくとも１つのオブジェクトの、抽象化通知で構成された動作仕様との

11

適合性を検証する第3 のコンピュータ・コード装置３１８を含めることができる。

【００３３】前述の機能をさらに、下記の例に関して例示する。ユーザが周辺入力３０６のキーボード上のキーを活動化したものと仮定する。この動作によって、キープレス事象がウィンドウ・ベースのプラットフォームからクライアント側のグラフィカル・ユーザ・インタフェースへ送られる。キープレス事象は、（１）事象の意味、（２）事象のフォーマットまたは構文の2 つの重要な態様を含む。キープレス事象は、プロセス間通信接続３０８を介してクライアント側へ転送される。

【００３４】クライアント側のオブジェクト指向グラフィカル・ユーザ・インタフェースによってキープレス事象が受取られると、事象ディスパッチ・オブジェクトはこの情報を見て、それを転送すべきかどうかを判定する。事象ディスパッチ・オブジェクトは次いで、判定に応じて情報を転送する。たとえば、キープレスが、文字「Ａ」キーと「シフト」キーを組み合わせて押すことに対応するものである場合、事象ディスパッチ・オブジェクトは、キープレス事象情報の受取時に、「Ａ」を表す情報をテキスト・フィールドに渡し、テキスト・フィールドは次いで、ディスプレイ上で「Ａ」の描画を実施する。すなわち、事象情報は、実行時交渉を介して抽象通知に関心を登録したオブジェクトへ転送できる抽象通知として表される。

【００３５】本発明の例示的な実施形態によれば、ツールキットは、ウィンドウ・ベースの処理系に必然的に存在する事象の動作仕様（すなわち、機能シグニチャ）のみを表すように抽象化された事象の通知に対して実施されている。たとえば、通常、事象にカプセル化され、それによって異なるウィンドウ・ベース・プラットフォームのために実施されるオブジェクト指向インタフェースへのその事象の移植性を制限していた意味および構文はなくなる。その代わり、事象は、特定のウィンドウ・ベース・システムに特有の処理系から独立するように抽象化される（すなわち、もはや最初のネイティブ・プラットフォームに特有のものではない）。（１）ネイティブ・ウィンドウ・ベース・プラットフォームおよび（２）ツール・キットが移植されたターゲット・ウィンドウ・ベース・プラットフォームに共通の通知の情報のみを使

12

用して、事象が認識されクライアント側のオブジェクトと相関付けされる。前述の例では、事象から共通の情報が抽出されることによって、単に、どの文字が押されたか（すなわち、「ａ」）を、通知をクライアント側のオブジェクトにマップするための動作仕様として示すことができる。

【００３６】事象の抽象化を例示するために、ボタンに関連付けられた事象（たとえば、キープレス事象）のコード化に関する簡略化された例を与える。この例で述べるコード・フラグメントは、当業者に周知の構造を表す。この例は、２つの異なる具体的なウィンドウ・システム事象の、単一の「抽象」表現としての抽象化と、その表現の、事象ソースとシンクとの間の形式的なプロトコルとしての表現を示すものである。この例では、プラットフォーム依存性をローカライズし、大部分のグラフィカル・ユーザ・インタフェース・フレームワークにプラットフォーム依存性を有させ、事象ソースとシンクとの間の形式的なプロトコルを作成し、それによってアプリケーションの継続期間中にソースとシンクとの間の関係を修正することができる有効な動的事象モデルを得る、本発明の実際の応用例が強調される。

【００３７】この例では、マウスまたはポインタ・ボタン・プレス事象の具体的なウィンドウ・システム事象通知が存在する、Ｘ Ｗｉｎｄｏｗ Ｓｙｓｔｅｍ™およびＮｅＸＴＳＴＥＰ™ Ｗｉｎｄｏｗシステムを考える。どちらのシステムでも、この事象通知は、プラットフォーム依存アプリケーション・プログラム・インタフェースおよびプロセス間通信機構を介して、ウィンドウ・システム・サーバから、システムからそのような通知を受取るアプリケーションとして選択されたクライアント・アプリケーションへ転送される。しかし、各システムは、構文と意味の両方に関して、そのような通知の意味的に異なる具体的表現を有する。たとえば、Ｘ Ｗｉｎｄｏｗ Ｓｙｓｔｅｍでは、事象は下記のＣタイプ定義および定数によって記述される（詳細については、Ｓｃｈｅｉｆｌｅｒ &Ｇｅｔｔｙｓ著「Ｔｈｅ Ｘ Ｗｉｎｄｏｗ Ｓｙｓｔｅｍ」（Ｄｉｇｉｔａｌ Ｐｒｅｓｓ、１９９２ＩＳＢＮ１－５５５５８－０８８－２）を参照されたい）。

```
typedef struct {
int type;               //==ボタンプレス
unsigned long serial;   //最後に処理した要求の通し番号
Bool      send_event;   //合成生成
Display*  display;      //ウィンドウ・サーバ・クライアント参照
Window    window;       //事象の転送先のウィンドウ
Window    root;         //画面のルート・ウィンドウ
Window    subwindow;    //事象が発生したウィンドウ
Time      time;         //サーバ・タイムスタンプ
int       x,            //画素ベース、原点左上
```

```
13                                                    14
        y;
int     x_root,     //画面関係のコード
        y_root;
unsigned int state;     //キーボード修飾子の状態
unsigned int button;    //どのボタンを押したか。
Bool    same_screen;
} XButtonEvent;
typedef XButtonEvent XButtonPressedEvent;
typedef XButtonEvent XButtonReleasedEvent;
/*上記の「ｔｙｐｅ」フィールドに関する定数*/
#define ButtonPress     4
/*上記の「ｓｔａｔｅ」フィールドに関する定数*/
#define ShiftMask       (1<<0)
#define LockMask        (1<<1)
#define ControlMask     (1<<2)
#define Mod1Mask        (1<<3)
#define Mod2Mask        (1<<4)
#define Mod3Mask        (1<<5)
#define Mod4Mask        (1<<6)
#define Mod5Mask        (1<<7)
/*上記の「ｂｕｔｔｏｎ」フィールドに関する定数*/
#define Button1         1
#define Button2         2
#define Button3         3
#define Button4         4
#define Button5         5
```

【００３８】ＮｅＸＴＳＴＥＰ　Ｗｉｎｄｏｗ　Ｓｙｓ　　ＥＰ　Ｇｅｎｅｒａｌ　Ｒｅｆｅｒｅｎｃｅ（ｖｏｌｕｍｅ２）」（１９９２ＩＳＢＮ０－２０１－６２２２１－２）を参照されたい）。ｔｅｍでは、Ｂｕｔｔｏｎ事象は下記のように定義される（詳細については、ＮｅＸＴ　Ｃｏｍｐｕｔｅｒ　Ｉｎｃ、Ａｄｄｉｓｏｎ　Ｗｅｓｌｅｙ著「ＮｅＸＴＳＴ

```
30
/*EventDataタイプ: 事象のデータ・フィールドを定義する*/
typedef union {
    struct {                            /*マウス事象の場合*/
        short   reserved;
        short   eventNum;               /*このボタンに関する一意の識
                                          別子*/
        int     click;                  /*この事象のクリック状態*/
        unsigned char  pressure;        /*圧力値: ０＝なし、２５５＝
                                          フル*/
        char    reserved1;
        short   reserved2;
    } mouse;
    /*話を明確にするために他の事象タイプ・データを削除する。*/
} NXEventData;
/*The event record!*/
    typedef struct {
        float x;
        float y;
    } NXPoint;
    typedef struct_NXEvent {
```

```
    int       type;             /*上記からの事象タイプ*/
    NXPoint   location;         /*ウィンドウ内のベース座標、
                                  左下から*/
    long      time;             /*開始時からの垂直間隔*/
    int       flags;            /*キー状態フラグ*/
    unsigned int   window;      /*割り当てられたウィンドウの
                                  ウィンドウ番号*/
    NXEventData   data;         /*タイプ依存データ*/
    DPSContext    ctxt;         /*事象が発生したコンテキスト
                                */
} NXEvent, *NXEventPtr;
/*上記の「ｔｙｐｅ」フィールドに関する定数*/
#define NX_LMOUSEDOWN    1      /*左マウス下降事象*/
#define NX_LMOUSEUP      2      /*左マウス上昇事象*/
#define NX_RMOUSEDOWN    3      /*右マウス下降事象*/
#define NX_RMOUSEUP      4      /*右マウス上昇事象*/
#define NX_MOUSEMOVED    5      /*マウス移動事象*/
#define NX_LMOUSEDRAGGED 6      /*左マウス・ドラッグ事象*/
#define NX_RMOUSEDRAGGED 7      /*右マウス・ドラッグ事象*/
#define NX_MOUSEDOWN     NX_LMOUSEDOWN    /*シノニム*/
#define NX_MOUSEUP       NX_LMOUSEUP      /*シノニム*/
#define NX_MOUSEDRAGGED NX_LMOUSEDRAGGED
                                 /*シノニム*/
/*上記の「ｆｌａｇｓ」フィールドに関する定数*/
#define   NX_ALPHASHIFTMASK      (1 << 16)/*アルファ・ロック
                                  がオンの場合*/
#define   NX_SHIFTMASK           (1 << 17)/*シフト・キーが押
                                  された場合*/
#define   NX_CONTROLMASK         (1 << 18)/*コントロール・キ
                                  ーが押された場合
                                */
#define   NX_ALTERNATEMASK       (1 << 19)/*ａｌｔキーが押さ
                                  れた場合*/
#define   NX_COMMANDMASK         (1 << 20)/*コマンド・キーが
                                  押された場合*/
#define   NX_NUMERICPADMASK      (1 << 21)/*数値パッド上のキ
                                  ーである場合*/
#define   NX_HELPMASK            (1 << 22)/*ヘルプ・キーが押
                                  された場合*/
/*装置依存ビット*/
#define   NX_NEXTCTRLKEYMASK     (1 << 0) /*コントロール・キ
                                  ー*/
#define   NX_NEXTLSHIFTKEYMASK   (1 << 1) /*左側シフト・キー
                                */
#define   NX_NEXTRSHIFTKEYMASK   (1 << 2) /*右側シフト・キー
                                */
#define   NX_NEXTLCMDKEYMASK     (1 << 3) /*左側コマンド・キ
                                  ー*/
#define   NX_NEXTRCMDKEYMASK     (1 << 4) /*右側コマンド・キ
                                  ー*/
```

15　　16

```
#define    NX_NEXTLALTKEYMASK      (1 << 5) /*左側a l t キー*/
#define    NX_NEXTRALTKEYMASK      (1 << 6) /*右側a l t キー*/
#define    NX_STYLUSPROXIMITYMASK (1 << 7) /*スタイラスが近接
                                            している場合( タ
                                            ブレット の場合)
                                           */
#define    NX_NONCOALSESCEDMASK    (1 << 8) /*事象合体がディス
                                            エーブルされた状
                                            態で事象が生成さ
                                            れた。*/
```

【００３９】上記の定義から、２つの処理系が具体的な構文および意味において著しく異なることは明らかである。したがって、これらの定義をネイティブ形で使用して、各ターゲット・ウィンドウ・システムごとの並列処理系を作成する包括的な条件付きコンパイル命令なしに両方のプラットフォーム間で移植できるグラフィカル・ユーザ・インタフェース・ツールキットの処理系を作成することはできない。しかし、本発明の例示的な実施形態によれば、両方のウィンドウ・システム間で移植できるボタン・プレスの共通の定義を抽象化し、プラットフォーム依存コードをツールキットの単一または比較的少数の構成要素としてローカライズすることによって、ツールキットを実施する大部分のコードにプラットフォーム依存性を有させることができる。

【００４０】上記で定義した具体的なウィンドウ・システム事象通知が与えられた場合、例示的な実施形態による処理系の第1のステップは、具体的な事象の共通の抽象を定義することである。この例では、オブジェクティブCプログラミング言語を使用する。なぜなら、これはＮｅＸＴＳＴＥＰや(Ｘ　Ｗｉｎｄｏｗ　Ｓｙｓｔｅｍ用の)ＯｐｅｎＳｔｅｐなどのプログラミング・システムで使用される処理系言語であるからである。抽象事象を使用して、すべての事象抽象に共通する事象タイプ、そのソース、その他の基本情報が定義される。

```
@interface AbstractEvent: Object
{
        enum {
            AEButtonPress,
            AEButtonRelease,
            AEMouseMoved,
            // ...
        }       eventType;
        SystemEventDispatcher    eventDispatcher;//プラットフォー
                                                 ム・ウィンドウ
                                                 ・システムから
                                                 事象をマップし
                                                 ディスパッチす
                                                 るオブジェクト
                                                 //
        unsigned int      eventSerial;//通し番号
        unsigned long     eventTimestamp;//システム・タイムスタン
                                         プ
}
// ...
@end
```

【００４１】次に、「ウィンドウ」に関連付けられた事象の概念、すなわち両方のターゲット・プラットフォームに共通の概念が導入される。

```
@interface AbstractWindowEvent: AbstractEvent
{
        unsigned int eventWindow;        //事象が発生したウィンド
                                         ウのＩＤ
}
```

17　　18

```
// ...
@end
```

【００４２】最後に、両方のターゲット・プラットフォームに関連する情報を含む、ボタン・プレス事象に関する特定の抽象が、具体的な事象の構文および意味とは異なる合成構文および意味を用いて定義される。

```
typedef enum {
      AKbdEShiftModifier = (1 < 0),
      AKbdECtrlModifier  = (1 < 1),
// ...
} AKbdEventModifiers;
@interface AbstractButtonPressEvent : AbstractWindowEvent
{
    float              eventX;
    float              eventY;
    enum {
        ABPELeftButton,
        ABPEMiddleButton,
        ABPERightButton
     }              eventButton;
        AKbdEventModifiers          eventKbdModifiers;
}
// ...
@end
```

【００４３】したがって、抽象ウィンドウ・システム事象オブジェクトの階層は、当業者なら認識可能であり、マウス・ボタン・プレス事象に関してＸ　Ｗｉｎｄｏｗ　ＳｙｓｔｅｍおよびｎｅＸＴＳＴＥＰ　Ｗｉｎｄｏｗ　Ｓｙｓｔｅｍの具体的な事象システムで使用される情報の合成を含むものとして定義されている。次に、任意のオブジェクト（オブザーバまたはシンク）が、特定のプロトコルまたはインタフェース仕様を介して他のオブジェクト（オブザービーまたはソース）からの通知を観測する際に関心を表すことができるようにするオブジェクト階層のインタフェースが定義される。

```
@interface Object(InterestProtocolRegistration)
- (Bool)addObserver:(Object *)observer
      forProtocol:(Protocol*)interestProtocol;
- (Bool) removeObserver:(Object *) observer
      forProtocol:(Protocol*)interestProtocol;
@end
```

【００４４】次いで、抽象事象定義に基づいて１組のプロトコルが定義される。これらのプロトコルは、上記で詳述した抽象事象記述で表されるシステムの状態の変化の通知を発行し受取ることができる任意のオブジェクト間の形式的なインタフェースを記述する。

```
@protocol AbstractEventObserverProtocol
-(void)  gotEvent:(AbstractEvent*)theEvent
        fromSource: (Object       *)theEventSource;
@end
@protocol AbstractWindowEventObserverProtocol
-(void)gotWindowEvent:(AbstractWindowEvent*)
theWindowEvent
   fromSource:(Object       *)theEventSource;
@end
@protocol AbstractButtonPressEventObserverProtocol
```

19　　20

```
- (void)gotButtonPressEvent:(AbstractButtonPressEvent*)
theWindowEvent
        fromSource:(Object                    *)
        theEventSource;
@end
```

【００４５】これらの抽象プロトコルを定義した後、これらの通知を受信したい「ＢｕｔｔｏｎＣｏｎｔｒｏ　＊1」オブジェクトを定義することができる。

```
@interface ButtonControl : Control
<AbstractButtonPressEventObserverProtocol,
AbstractWindowEventObserverProtocol,
                                        // ...
                                        >
// ...
-(ButtonControl)newButtonControl;
-(void)       dealloc;
-(void)gotButtonPressEvent:(AbstractButtonPressEvent*)
theWindowEvent
                fromSource:(Object                *)
                theEventSource;
-(void)gotWindowEvent:(AbstractWindowEvent*)
                theWindowEvent
                fromSource: (Object       *)
                theEventSource;
// ...
@end
```

【００４６】次いで、処理系の関連する部分が与えられる。

```
@implementation ButtonControl
-(ButtonControl) newButtonControl
{
```

【００４７】このメソッドは、ＢｕｔｔｏｎＣｏｎｔｒｏｌの新しいインスタンスを作成する。副作用として、ＢｕｔｔｏｎＣｏｎｔｒｏｌは、ＢｕｔｔｏｎＰｒｅｓ※

※s 事象の通知を受取るために、それ自体にツールキットの「ｅｖｅｎｔＤｉｓｐａｔｃｈｅｒ」オブジェクトを登録する。

```
self = [super init];
```

【００４８】これで、このオブジェクトはウィンドウ・システム事象ディスパッチャに登録され、プロトコル通知を受け取ることができるようになる。

```
[[SystemEventDispatcher eventDispatcher]
        addObserver: self
        forProtocol:
@protocol(AbstractButtonPressEventObserverProtocol)
        ];
        return(ButtonControl*)self;
}
-(void) dealloc
{
```

【００４９】このメソッドは、ＢｕｔｔｏｎＣｏｎｔｒｏｌのインスタンスを解放するために呼び出される。副作用として、ＢｕｔｔｏｎＣｏｎｔｒｏｌはそれ自体をｅｖｅｎｔＤｉｓｐａｔｃｈｅｒオブジェクトから削除する。

```
[[SystemEventDispatcher eventDispatcher]
        removeObserver: self
        forProtocol:
@protocol(AbstractButtonPressEventObserverProtocol)
];
```

```
    21                                                    22
    [super dealloc];
}
- (void) gotButtonPressEvent:(AbstractButtonPressEvent*)
theWindowEvent
            fromSource:(Object*              )
            theEventSource
{
   //このメソッドは、これに対してボタン・プレスが行われたときに呼び出
される
   //制御...
   //...次に、ButtonControl活動化処理を行う。...
}
@end
```

【００５０】グラフィカル・ユーザ・インタフェース・ツールキット・オブジェクトをこれらの抽象プロトコルで定義した後、「ＳｙｓｔｅｍＥｖｅｎｔ Ｄｉｓｐａｔｃｈｅｒ」クラスが実施される。本発明の例では、これ＊は、ウィンドウ・システム依存コードを含む、グラフィカル・ユーザ・インタフェース・ツールキットの事象サブシステム内の唯一のクラスである。

```
@interface SystemEventDispatcher : Object
+ (SystemEventDispatcher) eventDispatcher;
- (void) run;
- (Bool) addObserver:(Object *)observer
  forProtocol:(Protocol*)interestProtocol;
- (Bool) removeObserver:(Object *)observer
orProtocol:(Protocol*)interestProtocol;
-(void) dispatchAbstractEvent:(AbstractEvent*)absev;
@end
```

【００５１】すべてのプラットフォーム依存コードをカプセル化するＳｙｓｔｅｍＥｖｅｎｔ Ｄｉｓｐａｔｃｈｅｒのクラス専用インタフェースが定義される。

```
@interface SystemEventDispatcher(PlatformDependentStuff)
@private
#ifdef XWindows
-(XEvent)_nextEvent;   //XWindowサーバから次のXEventを取り込む
-(void)    _mapXEvent:(XEvent      *)xEvent
OntoAbstractEvent:(AbstractEvent**)returnEvent;
// ...
#endif
#ifder NeXTSTEP
-(NXEvent)_nextNXEvent; //NeXTSTEP DPSから次のNXEventを取り込む。
-(void)   _mapNXEvent:(NXEvent*      )nxEvent
   OntoAbstractEvent:(AbstractEvent**)returnEvent;
// ...
#endif
// ...
@end
@implementation SystemEventDispatcher
{
   // ...
  HashTable* buttonPressEventObservers;//ウィンドウIDでハッシュされた
                                       //ボタン・プレス事象に関心を持
                                       //つすべてのControlを含む。
```

```
    // ...
}
+ (SystemEventDispatcher)eventDispatcher
{
    //システム事象ディスパッチャの新しいインスタンスを作成する。
    static SystemEventDispatcher* eventDispatcher = nil;
    if (eventDispather = = nil) {
            eventDispatcher = [[SystemEventDispatcher
            alloc] init];
    }
    return eventDispatcher;
}
- run
{
  //事象ディスパッチャを実行し、システムから事象を取り込み、事象をプラ
  //ットフォーム特有の形式から抽象マッピングにマップし、その後、すべて
  //の適格なオブザーバにディスパッチする。 ...
  for(;;){
          AbstractEvent*     absev;
#ifdef XWindows
          [self     _mapXEvent:[self_nextXEvent]
          OntoAbstractEvent: <absev
          ];
#endif
#ifdef NeXSTSTEP
          [self     _mapNXEvent:[self_nextNXEvent]
                OntoAbstractEvent: <absev
          ];
#endif
          [self dispatchAbstractEvent: absev];
    }
 }
- (void) dispatchAbstractEvent:(AbstractEvent*)absev
{
    //抽象事象を適格なオブザーバにディスパッチする。
    switch([absev eventType] {
          // ...
          case AEButtonPress:
                //抽象事象のウィンドウIDに関連付けられたControlを見つ
                  け、事象をそのControlにディスパッチする。 ...
                Control* c = [buttonPressEventObserversvalueForKey:
                              [absev eventWindow]
                      ];
                [c gotButtonPressEvent: absev
                            FromSource: self
                      ];
                break;
                // ...
    }
}
```

```
- (Bool)addObserver: Object*  observer.
      forProtocol:Protocol* interestProtocol
{
              //特定のオブザーバ・プロトコルに関するオブザーバ( 適格
                な場合) を登録する。
              // ...
              if(interestProtocol = =)
  @protocol(AbstractButtonPressEventObserverProtocol) <<
              [observer conformsTo: interestProtocol]) {
                    //オブザーバがプロトコルに適合している場合、その
                    //ウィンドウ識別子で索引付けされたハッシュ・テー
                    //ブルにそのオブザーバを保存する。
                    [buttonPressEventObservers
                         insertkey:(const void*)[observer window];
                         value: (void     *)observer
                         ];
              }
              // ...
             return(Bool)True;
  }
  // ...
  @end
  @implementation
  SystemEventDispatcher(PlatformDependentStuff)
  #ifdef XWindows
  // ...
  - (void)     _mapXEvent:(XEvent*      )xEvent
       OntoAbstractEvent:(AbstractEvent**)returnEvent
{
                    //システム事象を事象抽象上にマップするためのプラ
                      ットフォーム依存コード
                    switch(xEvent->type){
                         // ...
                         case ButtonPress:{
                               XButtonPressedEvent* xPress =
                                    (XButtonPressedEvent*)xEvent;
                         //新しいAbstractButtonPressEventを作成する
                           。
                         *returnEvent = [AbstractButtonPressEvent
                         new];
                         //次に、XButtonPressedEventに存在するプラ
                         //ットフォーム依存情報に基づいてAbstract
                         //ButtonPressEventのメンバー変数を初期設定
                         //する。
                         // ...
                    }
                    break;
                   // ...
              }
```

23　　24

```
}
// ...
#endif
#ifdef NeXTSTEP
// ...
#endif
@end
```

【００５２】簡単な例示的な「メイン」プログラムによって、ＳｙｓｔｅｍＥｖｅｎｔ ＤｉｓｐａｔｃｈｅｒがＢｕｔｔｏｎＣｏｎｔｒｏｌを示す。

```
int main(const int argc, const char **const argv)
{
    [ButtonContrl newButtonCotnrol]; //ButtonControlを作成する
    [[SystemEventDispatcher eventDispatcher]run]; //ディスパッチャを実行する。
}
```

【００５３】本発明によれば、動作仕様の抽象通知は、事象ディスパッチ・オブジェクトと共に、図2のステップ２０８に関して説明したように、グラフィカル・ユーザ・インタフェース・ツールキット内の特定のオブジェクトが通知に適合しているかどうかを実行時交渉する能力を与える。本明細書では実行時交渉と呼ばれる実行時適合性試験を行う能力によって、ビュー・フィールド内のオブジェクトと制御フィールドまたはモデル・フィールド内のオブジェクトとの間の関係は、システムの実行継続期間中に動的に変化することができる。例示的な図３のシステムでは、実行時交渉によって、事象ディスパッチ・オブジェクト内の再マッピングを介して関係を動的に変更（すなわち、確立または確立解除）することができる。

【００５４】前述の機能、特に実行時交渉機能をより明確に例示するために、一方はウィンドウ・ベースのプラットフォーム内のオブジェクトであり、他方はグラフィカル・ユーザ・インタフェース内のオブジェクトである２つの匿名オブジェクトを考える。この2つのオブジェクトは、互いの存在のみを知り、他のことは何も知らない。この2つのオブジェクトを本明細書では、「ベンダ」および「顧客」と呼ぶ。ベンダ４０２および顧客４０４を図4に示す。

【００５５】ベンダの動作仕様が、キープレス事象など、顧客（たとえば、クライアント側アプリケーションの顧客）が使用したいプロトコルを含むと仮定すると、ベンダは、キープレス事象の何らかの通知を顧客にエクスポートできなければならない。顧客は、ベンダの存在を知り、ベンダに情報（たとえば、通知）を渡す能力を検証するために実行時交渉を開始することに関心を抱く。したがって、顧客は、ベンダが顧客の動作仕様に適合しているかどうかを知る必要がある。

【００５６】この開始位置が与えられた場合、実行時交渉は、顧客が、動作仕様「Ａ」（すなわち、顧客が使用したいプロトコル）を扱っているかどうかをベンダに問い合わせることによって開始される。ベンダが動作仕様「Ａ」を扱っていない場合、顧客はベンダとの情報転送を確立する試みを中断する。しかし、ベンダが動作仕様「Ａ」を取り扱っていることを示した場合、顧客は、「Ａ」に対応する動作仕様に関する「関心」をベンダに登録する。ベンダはこの動作仕様をサポートしているので、顧客は、この動作仕様に関連する状態変化が起こったときにベンダが顧客のビューに情報を転送できるようにベンダと共にこの動作仕様に参加したいことをベンダに知らせる。

【００５７】顧客が、状態変化に関する通知を受取るために動作仕様に参加し、ベンダと交渉したいことを確認した後、ベンダは、動作仕様「Ａ」に適合（すなわち、動作仕様「Ａ」を遵守）しているかどうかに関して顧客に問い合わせる。顧客が動作仕様「Ａ」に適合していると仮定すると、関係（すなわち、実行時契約）が確立され、この2つのオブジェクト間で抽象通知が流れることができる。

【００５８】例示的な実施形態によれば、システムも実行継続期間中の任意の時間に、ベンダまたは顧客、あるいはその両方はこの関係を取り消すことができる。すなわち、どちらのオブジェクトもそれ自体をこの関係から削除することができる。そのような削除が望ましいのはたとえば、顧客がジョブを完了しこの関係を放棄する場合である。どちらかのオブジェクトがこの関係を放棄する場合、前述のように新しい関係を確立することができる。

【００５９】例示的な実行時交渉について説明したが、次に、本発明によるオブジェクトどうしの例示的なマッピングについて述べる。これに関しては、図5を参照する。図5では、ウィンドウ・ベースのシステムのディスプレイ上のプッシュボタンの活動化に対応する事象を使用してある事象がトリガされる。例示的な実施形態によ

25

れば、この事象に対応する具体的な情報は、グラフィカル・ユーザ・インタフェースによって前述のようにウィンドウ・ベースのプラットフォームから抽象情報にマップされる。この場合、プッシュボタン５０２は、クライアント側オブジェクト指向グラフィカル・ユーザ・インタフェースで、モデル・ビュー・コントローラ・パラダイムのビューとコントローラの両方に対応するオブジェクトとして表される。このオブジェクトは、ビューとしては、ディスプレイ上にプッシュボタンを描画する責任を負う。コントローラとしては、プッシュボタンが押され、あるいは解放されたことを示すために発生した状態変化を記述する。

【００６０】プッシュボタンに関する動作仕様が、（１）「プッシュボタンが押された」と（２）「プッシュボタンが解放された」という２つのメッセージを生成するプロトコル「Ｂｕｔｔｏｎ　Ｃｌｉｃｋｓ」を含むと仮定する。プッシュボタンの場合、抽象化通知はさらに、プッシュボタンが押された時間（すなわち、タイム・スタンプ）、またはプッシュボタンを用いて活動化されたキーボード修飾子（たとえば、文字キーを用いたシフト・キーの活動化）、またはプッシュボタン活動化時のマウスの位置（たとえば、マウス上のボタンが活動化された場所）を識別する情報を含むことができる。さらに、事象ディスパッチ・オブジェクトが「Ｂｕｔｔｏｎ　Ｃｌｉｃｋｓ」プロトコルを扱っていると仮定する。実行時交渉時に、プッシュボタンは、事象ディスパッチ・オブジェクトに「Ｂｕｔｔｏｎ　Ｃｌｉｃｋｓ」プロトコルを扱っているかどうかを問い合わせる。事象ディスパッチ・オブジェクト５０４がこのプロトコルを扱っている場合、プッシュボタンは、事象ディスパッチ・オブジェクトと共に「Ｂｕｔｔｏｎ　Ｃｌｉｃｋｓ」プロトコルに参加したいことを示すことによって応答する。したがって、プッシュボタンは「Ｂｕｔｔｏｎ　Ｃｌｉｃｋｓ」プロトコルに「関心」を登録する。

【００６１】事象ディスパッチ・オブジェクトは次に、プッシュボタンに「ＢｕｔｔｏｎＣｌｉｃｋｓ」プロトコルに適合しているかどうかを問い合わせる。プッシュボタンが肯定応答した場合、関係が確立される。そのため、事象ディスパッチ・オブジェクトは、ウィンドウ・ベースのプラットフォーム５０６からボタンプレス事象を受取ると、プッシュボタンに「プッシュボタン・ディプレス」事象を通知する。

【００６２】本発明の例示的な実施形態は事象を抽象化するので、もちろん、処理系固有のある種の情報は失われることがある。しかし、情報が失われる代わりに、システムの移植性が向上する。さらに、当業者には理解されるように、処理系固有の情報は、このプラットフォーム特有の処理系用のクライアント側オブジェクト指向グラフィカル・ユーザ・インタフェースにおいて情報を個別にコード化することによって保持することができる。

26

【００６３】再び図5を参照すると分かるように、当業者には、前述のように通知を抽象化すると、オブジェクト間の１：１関係を維持する必要がなくなることが理解されよう。その代わり、抽象通知に「関心」を持つすべてのオブジェクトを通知を受取るようにマップすることができる。たとえば、抽象通知がタイマ５０８の出力に対応する場合、関連するプロトコル「ティック」を確立し、１秒経過するたびに事象ディスパッチ・オブジェクトに通知することができる。事象ディスパッチ・オブジェクトと、時間の追跡に「関心」を持つすべてのオブジェクト（たとえば、オブジェクト５１０および５１２）との間に独立の「Ｔｉｍｅｆｌｏｗ」プロトコルを確立することができる。そのようなオブジェクトは、たとえば、ディスプレイ上のクロックや、株式市場ティッカ・フィードや、タイマなどを含むことができる。したがって、時間の追跡に関心を持つすべてのオブジェクトのビューに、「Ｔｉｍｅｆｌｏｗ」プロトコルを介して時間の変化が通知される。したがって、「ティック」プロトコルが事象ディスパッチ・オブジェクトと１：１の関係を有する場合でも、「Ｔｉｍｅｆｌｏｗ」プロトコルは、事象ディスパッチ・オブジェクトと、時間の追跡に関心を持つすべてのオブジェクトとの関係を表す。

【００６４】当業者には、本発明の例示的な実施形態が多数の異なる方法で実施できることが理解されよう。たとえば、例示的な実施形態を使用して、グラフィカル・ユーザ・インタフェースの機能を、ソフトウェアとハードウェアのどちらかで実施されるウィンドウ・ベースのシステムに移植することができる。さらに、例示的な実施形態をクライアント・サーバ・システムに関して説明したが、当業者には、本発明を使用して、ウィンドウ・ベースのシステムがどこに存在するかにかかわらずにグラフィカル・ユーザ・インタフェースのツールキットをウィンドウ・ベースのシステムに移植できることが理解されよう。

【００６５】当業者には、本発明の趣旨または必須の特性から逸脱せずに本発明を他の特定の形態で実施できることが理解されよう。したがって、現在開示している実施形態はすべての点で例示的なものとみなされ、制限されない。本発明の範囲は、上記の説明ではなく添付の特許請求の範囲によって示され、本発明の意味および範囲ならびに等価物内のすべての変更は本発明に包含されるものとする。

【図面の簡単な説明】

【図1】　モデル・ビュー・コントローラ・パラダイムの抽象表現を示す図（Ａ）と図1Ａに示した抽象表現の例示的な応用例を示す図（Ｂ）である。

【図2】　本発明の例示的な実施形態による、ユーザ・インタフェースの、第1のウィンドウ・ベース・プラットフォームから第2のウィンドウ・ベース・プラットフォームへの移植に関する例示的なフローチャートであ

27

る。

【図3】　クライアント側インタフェースを第1のウィンドウ・ベース・プラットフォームから第2のウィンドウ・ベース・プラットフォームに移植するために本発明によって構成された例示的なシステムを示す図である。

【図4】　本発明の形態を示す図である。

【図5】　本発明によって構成されたシステムの、所与の動作仕様に適合しているオブジェクトの任意のインスタンスと、同じ動作仕様に適合している他のオブジェクト

28

との関係を動的に確立する能力を示す図である。

【符号の説明】

102　モデル

104、116　ビュー

106　コントローラ

108　矢印

112　スクロール・バー・オブジェクト

114　テキスト・ファイル

【図1】

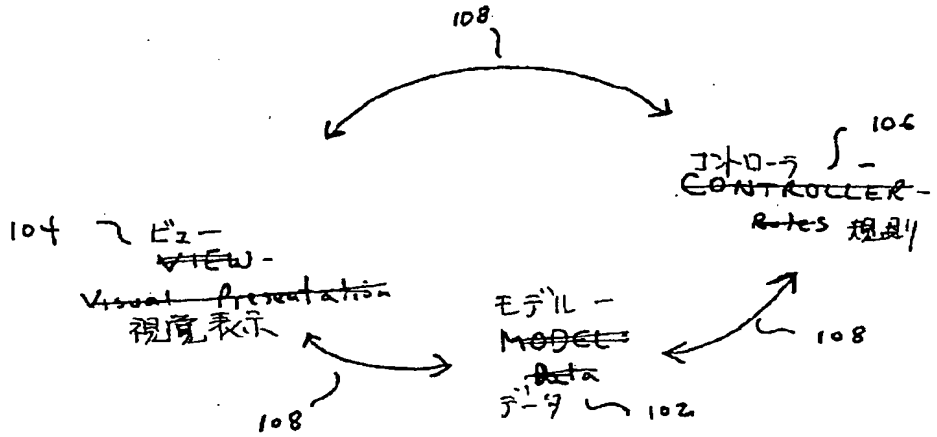


FIGURE 1A　A

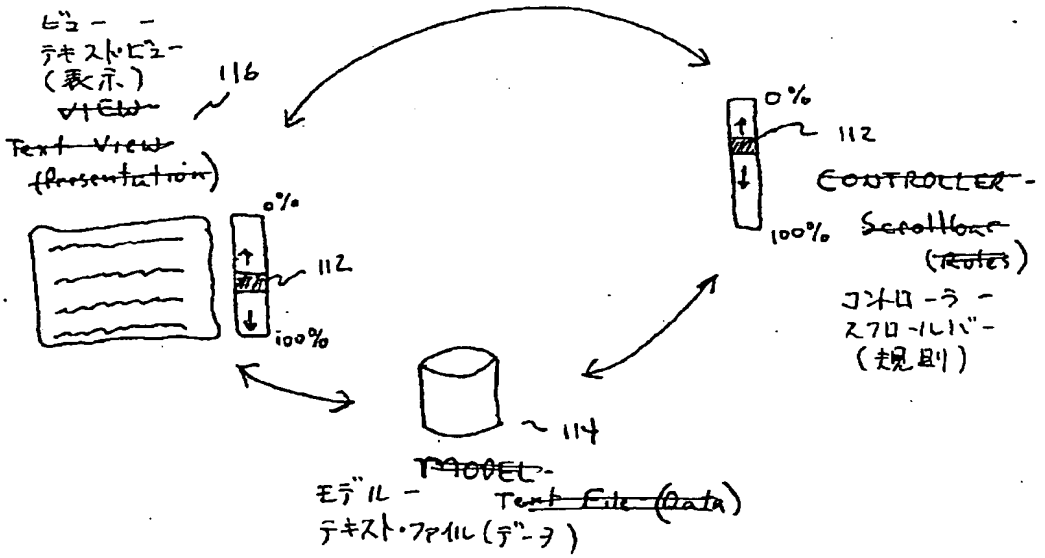
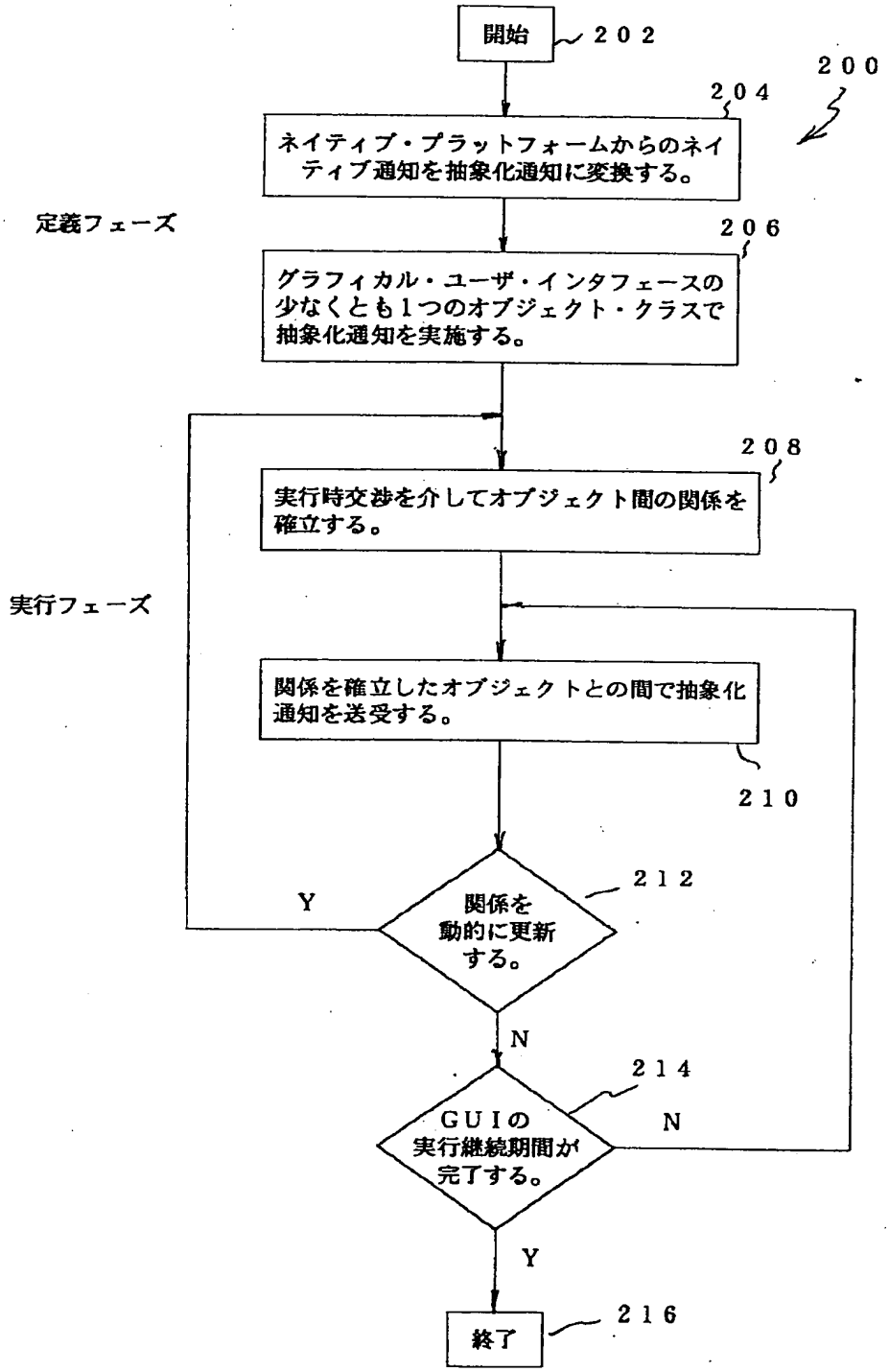
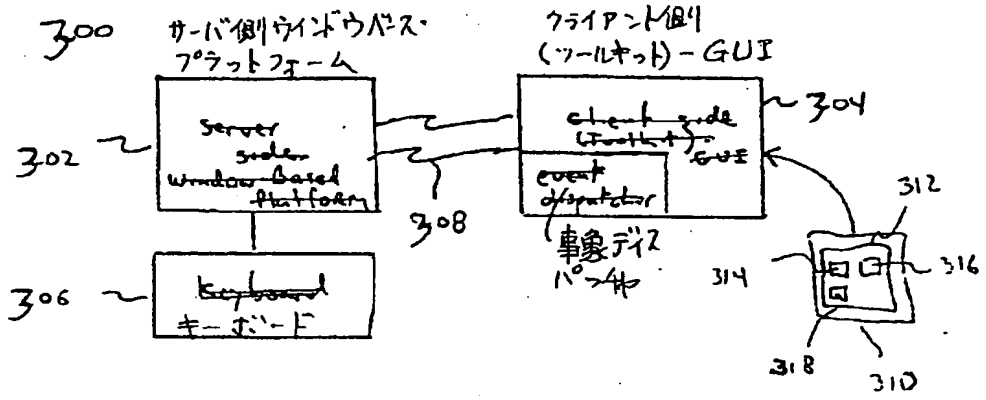


FIGURE 1B　B

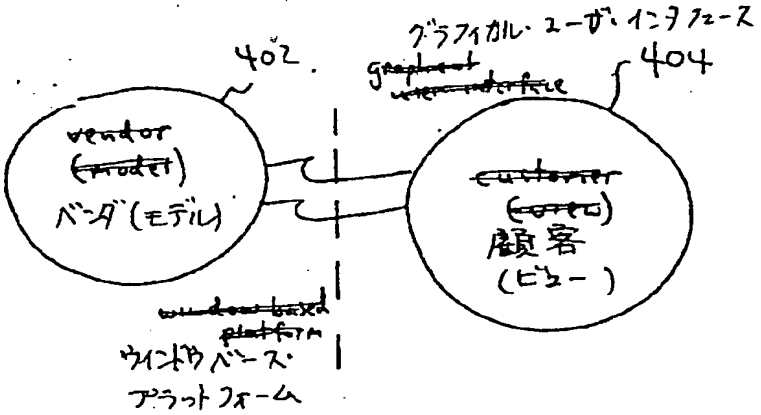
【 図2 】


開始
202
200
204
ネイティブ・プラットフォームからのネイティブ通知を抽象化通知に変換する。
定義フェーズ
206
グラフィカル・ユーザ・インタフェースの少なくとも1つのオブジェクト・クラスで抽象化通知を実施する。
208
実行時交渉を介してオブジェクト間の関係を確立する。
実行フェーズ
関係を確立したオブジェクトとの間で抽象化通知を送受する。
210
212
関係を動的に更新する。
Y
N
214
GUIの実行継続期間が完了する。
N
Y
終了
216

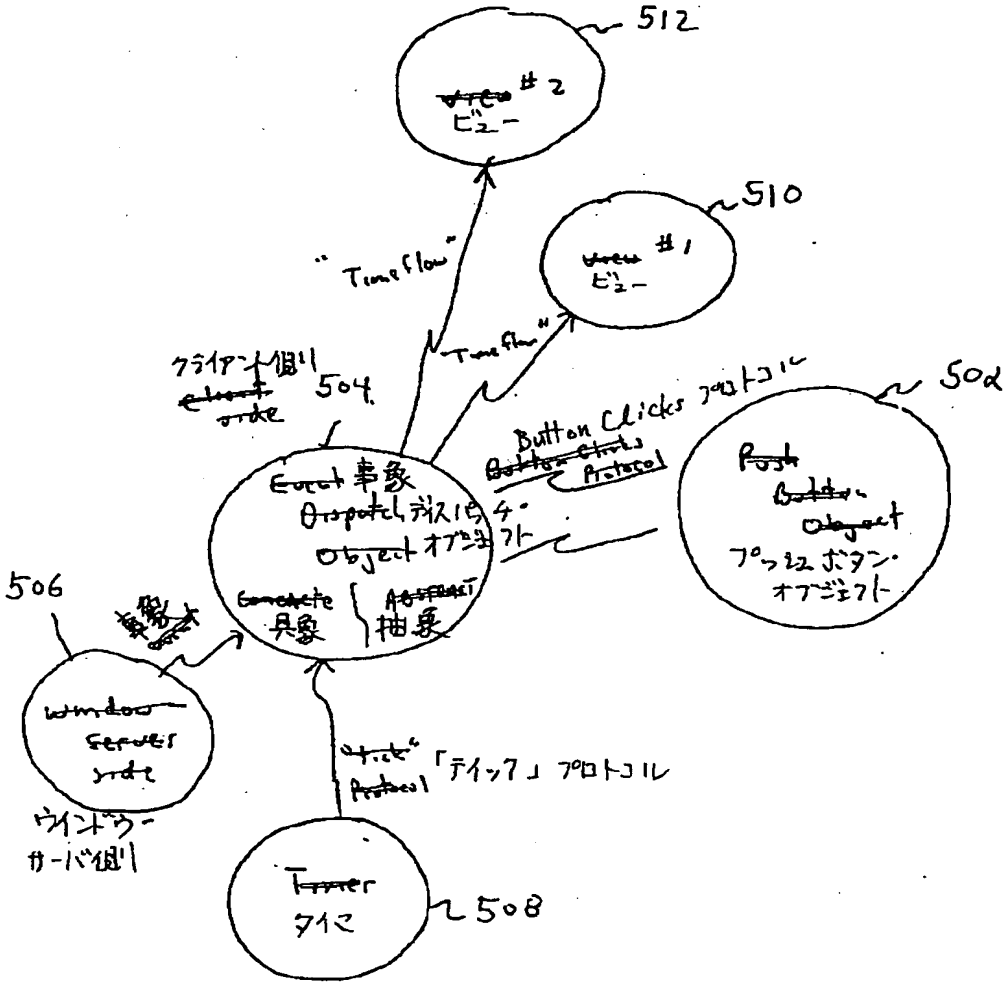
【 図3 】



【 図4 】

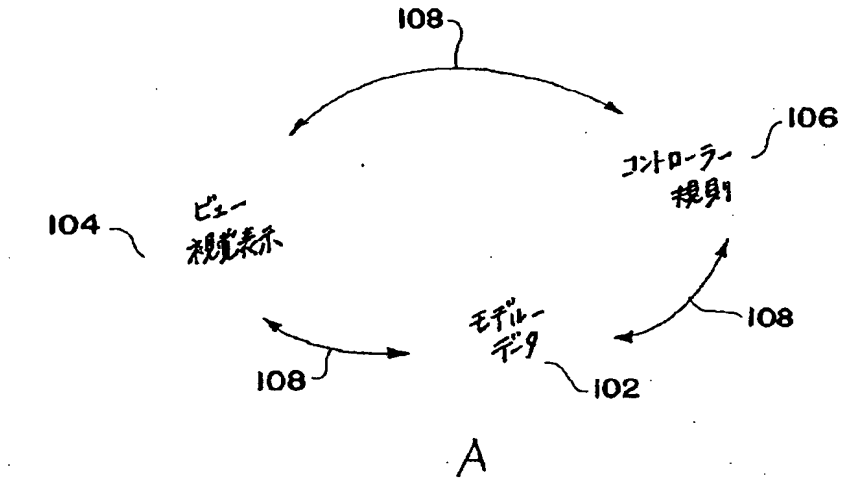


【 図5 】

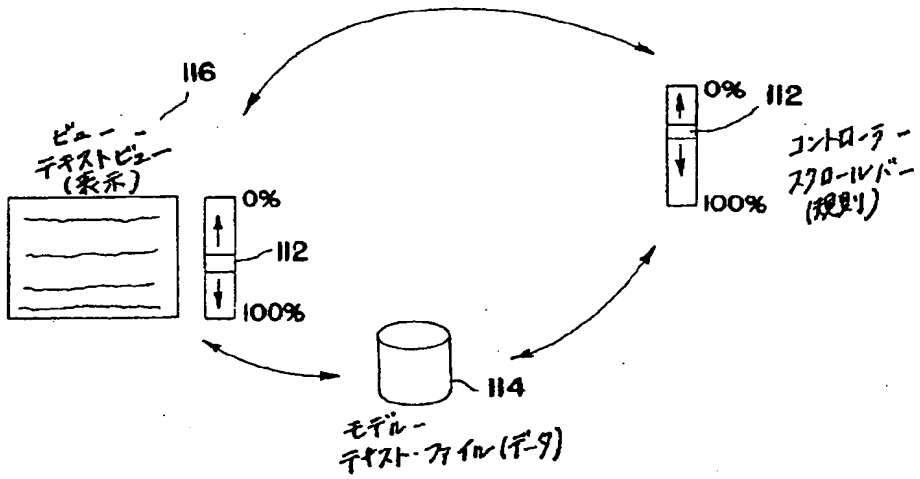


【 手続補正書】
【 提出日】平成9 年１１ 月2１ 日
【 手続補正1 】
【 補正対象書類名】図面
【 補正対象項目名】全図
【 補正方法】変更
【 補正内容】

【 図1 】

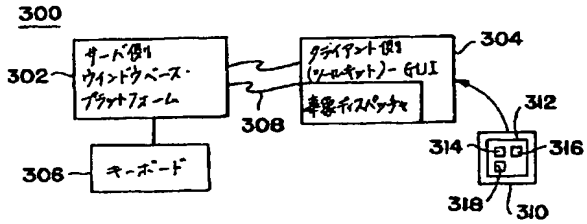


A

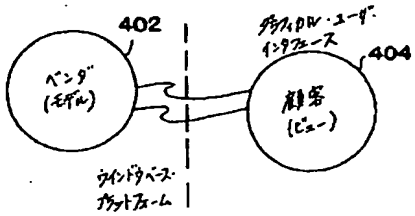


B

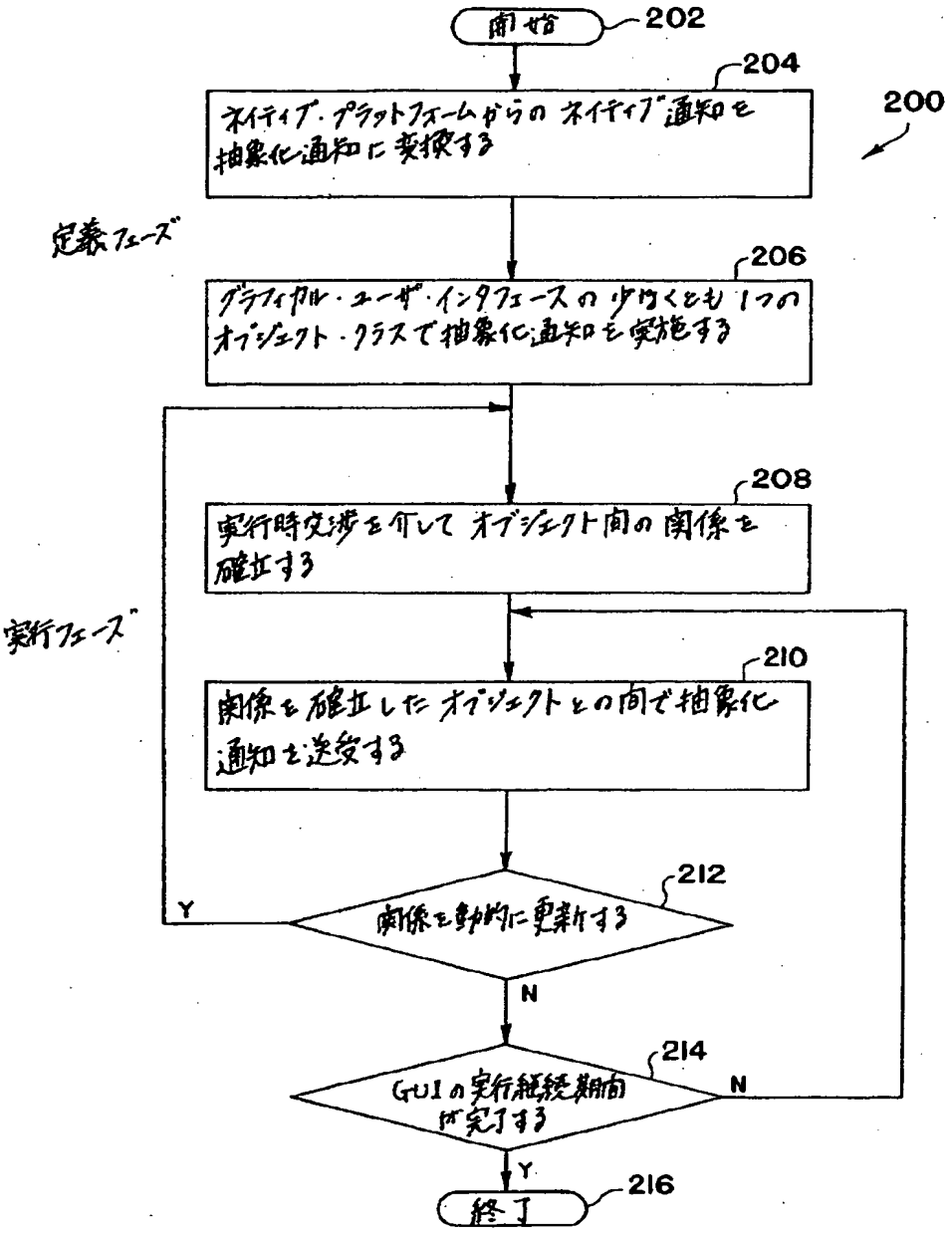
【 図3 】



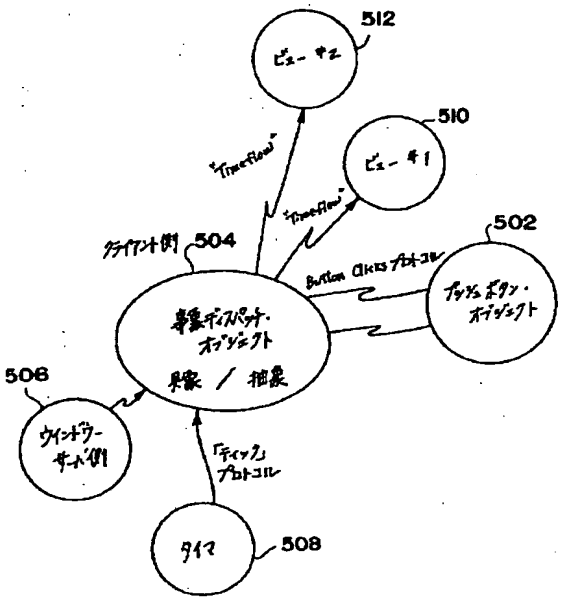
【 図4 】



【 図2 】


開始
202
204
ネイティブ・プラットフォームからのネイティブ通知を抽象化通知に変換する
200
定義フェーズ
206
グラフィカル・ユーザ・インタフェースの少なくとも1つのオブジェクト・クラスで抽象化通知を実施する
208
実行時交渉を介してオブジェクト間の関係を確立する
実行フェーズ
210
関係を確立したオブジェクトとの間で抽象化通知を送受する
212
関係を動的に更新する
Y
N
214
GUIの実行経験期間が完了する
N
Y
216
終了

【 図5 】



フロント ページの続き

(71)出願人　591064003
901 SAN ANTONI O ROAD
PALO ALTO, CA 94303, U.
S. A.